

Multicast Group Communication for CORBA*

L. E. Moser, P. M. Melliar-Smith,
P. Narasimhan, R. R. Koch and K. Berket

Department of Electrical and Computer Engineering
University of California, Santa Barbara, CA 93106
{moser, pmms, priya, ruppert, karlo}@alpha.ece.ucsb.edu

Abstract

Multicast group communication is a useful augmentation to CORBA both for fault-tolerant and highly available applications and for groupware and cooperative work applications. However, different multicast group communication protocols are appropriate in different environments, e.g., local area vs. wide area networks, and Internet vs. ATM. In this paper we present a multicast group communication engine and bridge for CORBA that allows different multicast group communication protocols to cooperate. The group communication engine places Lamport timestamps on messages, and multicasts messages to object groups using one or more group communication protocols. The group communication protocols reliably deliver the timestamped messages in timestamp order to the group communication engine, which integrates these streams of messages into a single stream for delivery in timestamp order.

1 Introduction

Fault tolerance and high availability can be provided for the Common Object Request Broker Architecture (CORBA) [17] by means of object replication, where the replicas of an object form an object group. However, object replication is of little value unless the states of the replicas of the objects remain consistent as those states change when methods are invoked on the object. Groupware and cooperative work applications based on CORBA also depend on object replication. Such applications require not only that the states of the replicas remain consistent but also that the activities of the applications are coordinated.

Group communication protocols [13] facilitate the maintenance of replica consistency, and the coordination of activities, by multicasting Request and Reply messages containing method invocations and responses, and by de-

livering the messages reliably in the same order to all of the members of a group. However, different group communication protocols are appropriate for effective operation in different environments. One protocol is effective for a local-area network, while a different protocol is appropriate for the Internet. The most appropriate protocol for an ATM-based private virtual network would be different again. Indeed, large-scale systems must be expected to operate with several different protocols simultaneously. Moreover, different protocols may operate in different domains with gateways between unreplicated objects and the domains, as shown in Figure 1.

Typical multicast group communication protocols are self-contained and make little or no provision for cooperation with other group communication protocols. Indeed, given the demands placed on them, this closed view is understandable and almost inevitable. Thus, what is needed for CORBA is a multicast group communication engine and bridge that allow different group communication protocols to be used concurrently.

In this paper we define the Multicast Group Internet Inter-ORB Protocol (MGIOP), which maps CORBA's General Inter-ORB Protocol (GIOP) specifications onto a multicast group communication protocol. We also present interfaces for a multicast group communication engine and bridge that allow multiple group communication protocols to cooperate. We do not constrain the internal algorithms of the group communication protocols or the message formats

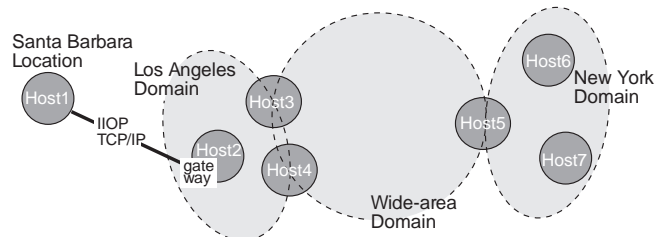


Figure 1: A gateway and domains.

*This research has been supported by the Defense Advanced Research Projects Agency and by the Office of Naval Research and the Air Force Research Laboratory, Rome, under Contracts N00174-95-K-0083 and F3602-97-1-0248, respectively.

that they use on the wire. The MGIOP engine and protocol provide the following services:

- Multicasting
- Reliable message delivery
- Causal message delivery
- Totally ordered message delivery
- Group membership
- Virtual synchrony.

First we briefly discuss CORBA's protocol specifications -- the General Inter-ORB Protocol (GIOP) and the Internet Inter-ORB Protocol (IIOP), the mapping of GIOP to TCP/IP.

2 GIOP and IIOP

The Generalized Inter-ORB Protocol (GIOP), defined for CORBA [17], makes several assumptions about the underlying transport protocol:

- The transport is connection-oriented. The connections provide the context within which request identifiers are unique.
- The transport is reliable in that bytes are delivered in the order in which they are sent and at most once.
- The transport can be viewed as a byte stream, without message size limitations, fragmentation, or alignments.
- The transport provides notification of disorderly connection loss, in particular, if the peer process aborts, the peer host crashes, or the network becomes disconnected.
- The transport's model for initiating connections can be mapped onto the connection model of TCP/IP. In particular, a server publishes a known network address in an Interoperable Object Reference (IOR), which the client uses when initiating a connection.

In the `MessageHeader_1_1` for GIOP version 1.1, shown in Figure 2, the `magic` field contains the characters "GIOP" and the `GIOP_version` field is the version of GIOP being used. The `flags` field specifies the byte ordering in subsequent elements of the message, where 0 indicates big-endian and 1 indicates little-endian. The `message_type` field specifies the type of message (Request, Reply, CancelRequest, LocateRequest, LocateReply, CloseConnection, MessageError, Fragment), corresponding to the values of the `MsgType_1_1` enumeration type. The `message_size` field specifies the number of octets in the message following the message header.

```
module GIOP {
    struct Version {
        octet major;
        octet minor;
    };

    enum MsgType_1_1 {
        Request, Reply, CancelRequest,
        LocateRequest, LocateReply,
        CloseConnection, MessageError,
        Fragment
    };

    enum MessageHeader_1_1 {
        char magic [4]; // GIOP
        Version GIOP_version;
        octet flags;
        octet message_type;
        unsigned long message_size;
    };
};
```

Figure 2: IDL specification of GIOP version 1.1.

```
module IIOP {
    struct Version {
        octet major;
        octet minor;
    };

    enum ProfileBody_1_1 {
        Version iiop_version;
        string host;
        unsigned short port;
        sequence <octet> object_key;
        sequence <IOP::TaggedComponent> components;
    };
};
```

Figure 3: IDL specification of IIOP version 1.1.

The Internet Inter-ORB Protocol (IIOP) is the TCP/IP instantiation of GIOP. Servers publish TCP/IP addresses in IORs, as defined by the `ProfileBody_1_1` in Figure 3. The `host` field in the `ProfileBody_1_1` identifies the Internet host to which the GIOP messages for the specified object are sent and the `port` field contains the port number at that host. The `object_key` is an opaque value supplied by the server that is used in the clients' Request messages to identify the server. The server maps the value onto the corresponding object when routing requests internally. The `components` field contains additional information that may be used when a client invokes the server.

Servers listen for connection requests. A client initiates a connection with a server, using the server's address given in the IOR. Once a connection is established, the client may send Request, LocateRequest or CancelRequest

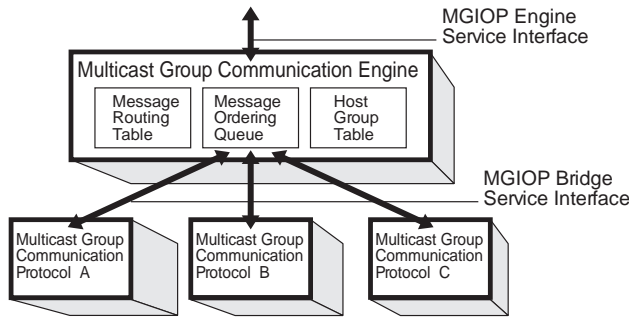


Figure 4: MGIOP Engine and Bridge.

messages over this connection. The server may send Reply, LocateReply and CloseConnection messages to the client over the same connection.

When a client sends a Request message, it includes a request identifier in the header of the message. The client subsequently uses the request identifier associate a Reply message with the Request.

IIOP provides only point-to-point communication between a single source and a single destination, whereas what is needed for simultaneous communication to the members of a group (as in the replicas of an object) is a multicast group communication protocol.

3 MGIOP Engine and Bridge

The multicast group communication engine and bridge, described here, enable multiple group communication protocols to be used concurrently, as shown in Figure 4. They coordinate, but do not control, the various group communication protocols. No forwarding or conversion from one multicast group communication protocol to another is necessary, and the group communication protocols never communicate directly with one another or know of each others' existence.

Each message multicast to an object group is conveyed by the same group communication protocol to all members of the group. However, a source can use different group communication protocols to reach different destination groups, and different sources can use different group communication protocols to reach the same destination group.

The multicast group communication engine depends on timestamps for message ordering and on heartbeat (*i.e.*, null) messages for liveness. Timestamps and heartbeat messages provide a convenient mechanism by which multiple group communication protocols can coexist without constraining the on-the-wire message formats and internal algorithms of the protocols. Existing multicast group communication protocols can easily be augmented with timestamps and heartbeat messages, if they do not employ those mechanisms.

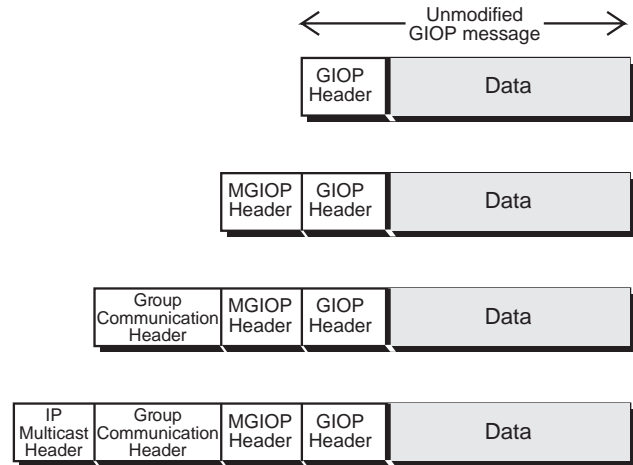


Figure 5: Successive encapsulations of a GIOP message.

Each multicast group communication protocol reliably delivers its timestamped messages in timestamp order to the multicast group communication engine. The group communication engine then integrates these streams of messages into a single stream of messages for delivery in timestamp order. The timestamps must be at least Lamport synchronized [10], but better performance can be achieved by means of synchronized clocks.

4 MGIOP Specification

Each GIOP message is encapsulated in a MGIOP header which, in turn, is encapsulated in a group communication message header, as shown in Figure 5. The MGIOP header comprises the MGIOP tag, MGIOP_version, source_domain_id, source_object_group_id, dest_domain_id, dest_object_group_id and connection_id fields, as shown in Figure 6. The MGIOP module, shown in Figures 7, 8 and 9, comprises the MGIOP Engine and MGIOP Bridge interfaces.

The MGIOP_tag consists of the characters MGIOP, and the MGIOP_version is the version of MGIOP being used. The source_object_group_id is the identifier of the object group that originated the message. The domain_id is the identifier of a domain that provides a context within which the source_domain_id is unique, and similarly for the dest_object_group_id and dest_domain_id. The connection_id consists of the domain identifier and object group identifier of the client object group and a connection number supplied by the client object group. It provides a context

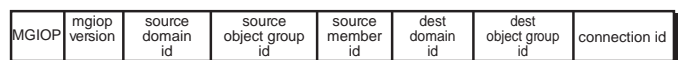


Figure 6: MGIOP header format.

```

module MGIOP {
  struct Version {
    octet major;
    octet minor;
  };

  typedef sequence<octet> DomainId;
  typedef long long ObjectGroupId;

  typedef sequence<octet> HostId;
  typedef sequence<HostId> HostIds;
  typedef long long HostGroupId;

  typedef unsigned long long Timestamp;
  typedef unsigned long TimeInterval;

  typedef unsigned long ConnectionNum;

  struct ConnectionId {
    DomainId domain_id;
    ObjectGroupId ogid;
    ConnectionNum num;
  };

  struct MGIOPSpecialTaggedProfile_1_0 {
    Version MGIOP_version;
    DomainId domain_id;
    ObjectGroupId object_group_id;
  };

  struct MGIOPMessageHeader_1_0 {
    char magic[5]; // i.e. MGIOP;
    Version MGIOP_version;
    DomainId source_domain_id;
    ObjectGroupId source_object_group_id;
    DomainId dest_domain_id;
    ObjectGroupId dest_object_group_id;
    ConnectionId connection_id;
  };

  struct MGIOPMessage_1_0 {
    MGIOPMessageHeader_1_0;
    GIOPMessage;
  };

  exception UnsupportedProtocol {};
  exception DomainNotFound {};
  exception ObjectGroupNotFound {};
  exception ConnectionNotFound {};
  exception HostGroupNotFound {};
  exception HostNotFound {};
  exception InvalidMessage {};
  exception InvalidTimestamp {};
  exception InvalidTimeInterval {};
}

```

Figure 7: IDL specification of the MGIOP module.

```

interface GroupMapper {
  void map_object_group(in ObjectGroupId object_group_id,
                        in HostIds host_ids);

  void delete_object_group(in ObjectGroupId object_group_id)
    raises(ObjectGroupNotFound);

  void add_host_for_object_group(
    in ObjectGroupId object_group_id,
    in HostId host_id)
    raises(ObjectGroupNotFound);

  void remove_host_for_object_group(
    in ObjectGroupId object_group_id,
    in HostId host_id)
    raises(ObjectGroupNotFound, HostNotFound);
};

interface ProtocolMapper {
  void map_protocol(in ProtocolTag tag,
                  in HostIds host_ids);
    raises(UnsupportedProtocol);
};

interface MGIOPEngine: GroupMapper, ProtocolMapper { };

```

Figure 8: IDL specification of the MGIOPEngine.

```

interface HostGroupManager {
  void add_host_to_host_group(in HostId host_id)
    in HostGroupId host_group_id
    raises(HostGroupNotFound);

  void remove_host_from_host_group(
    in HostId host_id,
    in HostGroupId host_group_id)
    raises(HostNotFound, HostGroupNotFound);
};

interface MessageHandler {
  void send_message(in MGIOPMessage msg,
                  in HostGroupId host_group_id,
                  in Timestamp timestamp)
    raises(InvalidMessage, HostGroupNotFound, InvalidTimestamp);

  Timestamp force();

  void set_heartbeat_interval(in TimeInterval heartbeat_interval)
    raises(InvalidTimeInterval);

  boolean is_message_available();

  MGIOPMessage deliver_message(out Timestamp timestamp);
};

interface MGIOPBridge: HostGroupManager, MessageHandler { };

```

Figure 9: IDL specification of the MGIOPBridge.

within which the standard CORBA request identifiers are unique. The MGIOP Engine at the destination (and also at the source) uses the connection_ids and request identifiers to detect and suppress duplicate invocations and duplicate responses from the replicas of an object.

The MGIOP Engine interface extends the GroupMapper and ProtocolMapper interfaces. The GroupMapper maps an object group to a set of hosts, and also adds hosts to, and removes hosts from, sets of hosts. The ProtocolMapper maps a group communication protocol to a set of hosts that can be reached using that protocol.

The MGIOPBridge interface extends the HostGroupManager and MessageHandler interfaces. The HostGroupManager adds hosts to, and removes hosts from, host groups. The MessageHandler multicasts and delivers messages to a host group that supports both the source and destination object groups. The MGIOP Engine invokes the methods of the HostGroupManager on the group communication protocols.

The send_message(), force() and set_heartbeat_interval() methods are invoked by the MGIOP Engine on the group communication protocol at the source. The force method is necessary to maintain Lamport causality [10] and total ordering when multiple group communication protocols are used, and one of those protocols changes the timestamp that the MGIOP Engine gives to the message; more details are provided in Section 6. The set_heartbeat_interval() method is used to specify the time between sending heartbeat messages, which are required for liveness.

The is_message_available() and deliver_message() methods are invoked by the MGIOP Engine on the group communication protocol at the destination. The is_message_available() method enables the MGIOP Engine to determine whether the group communication protocol has a message available to deliver to it.

5 MGIOP Semantics

The MGIOP Engine and protocols satisfy the following (informal) requirements:

- **Multicasting.** Messages are originated by an object within a source object group and are addressed to a destination object group. Messages are delivered to the members of both the source group and the destination group except that, if a group communication protocol does not service a group, it is not required to deliver messages to members of that group.

Multicasts may be simulated, for example by multiple TCP/IP connections, or may employ properties of the physical medium (e.g., hardware multicasts) for improved performance.

- **Reliable delivery.** Every message addressed to an object group or originated by an object group is delivered

to every member of the group, except for members that are suspected of being faulty.

- **Causal order.** The *precedes* relation is the transitive closure of:

- If object replica O sends message m1 before message m2, then m1 precedes m2.
- If message m1 is delivered to object replica O before O sends message m2, then m1 precedes m2.

If both m1 and m2 are delivered to object replica O, and m1 precedes m2, then m1 is delivered to O before m2 is delivered to O.

- **Total order.** The *ordered before* relation is the transitive closure of:

- If message m1 is delivered to object replica O before message m2 is delivered to O, then m1 is ordered before m2.
- If message m1 precedes message m2, then m1 is ordered before m2.
- The ordered before relation is acyclic.

If both m1 and m2 are delivered to object replica O, and m1 is ordered before m2, then m1 is delivered to O before m2 is delivered to O.

- **Group membership.** Membership changes occur at specific points in the sequence of messages. At any point in the sequence, object replicas O and Q in the same object group G have the same view of the group, i.e., “see” the same membership set.
- **Virtual synchrony.** If object replicas O and Q are in the same object group view G and transition together to the next group view G', then the same messages are delivered to O and Q while they are members of object group view G.

Virtual synchrony is used to ensure that a state transfer to initialize a new member of object group view G occurs at the point in the message order corresponding to a membership change. Thus, at the start of the next object group view G', all of the members of the group have the same state.

These properties are not provided by IIOP.

6 Connection Management

6.1 Addressing Replicas

As a concrete implementation of the GIOP specifications, MGIOP is connection-oriented. The MGIOP Engine uses a Multiprofile IOR that contains a Special Tagged Profile as the first profile, which is used to address the object group. The other profiles of the Multiprofile IOR are the profiles of

primary or backup gateways into the domain for the object group. The Special Tagged Profile contains the domain_id and the object_group_id of the destination object group.

6.2 Connection Establishment

The IORs that are provided for IIOP are sufficient for individual object addressing, but are insufficient for object group addressing. Fortunately, CORBA allows multiple profiles (address endpoints) to be inserted into a single IOR, and this can be exploited to enumerate the addresses of the members of an object group.

1. The client-side MGIOPEngine extracts the server object_group_id and domain_id from the Special Tagged Profile of the server's multiprofile IOR.
2. The client-side MGIOPEngine determines the set of hosts that support the client object group and the set of hosts that support the server object group, and forms the union of these two sets of hosts.
3. The client-side MGIOPEngine determines whether there exists a protocol and a host group that supports this union. If there does not exist such a protocol, then it raises an exception. If there exists such a protocol but there does not exist a host group that encompasses the union, the MGIOPEngine chooses a host group identifier and forms a host group with that identifier, using the add_host_to_host_group() method of the MGIOBridge repeatedly.
4. The client-side MGIOPEngine then chooses a connection number and opens a connection.

6.3 Connection Release

1. Like other GIOP messages, the CloseConnection message is multicast as a MGIOPEngine message. The Cancel-Request message is ignored because it can lead to inconsistencies in the states of the replicas of an object. Because the behavior of the replicas of an object must be deterministic, a connection is closed at the same logical point in time at the client and server replicas.

7 MGIOPEngine Information

7.1 Message Routing Table

The MGIOPEngine maintains the Message Routing Table using the information in its Object/Host Group Table, as shown in Figure 10. It updates the connection_id in the Message Routing Table when it establishes and releases a connection.

The MGIOPEngine consults the Message Routing Table to determine the appropriate set of hosts to which to

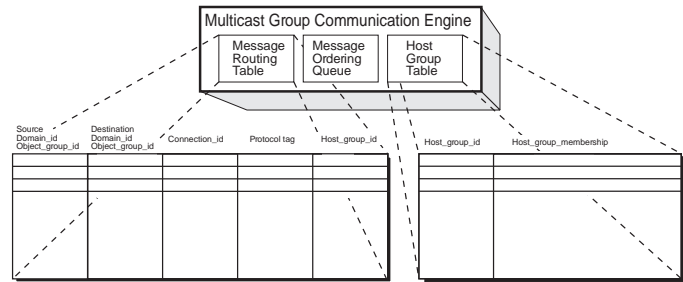


Figure 10: Message Routing Table and Object/Host Group Table.

multicast the message and the appropriate multicast group communication protocol to use.

7.2 Message Ordering Queue

Each multicast group communication protocol delivers a sequence of messages to the MGIOPEngine in timestamp order. The MGIOPEngine inserts these messages into its Message Ordering Queue and delivers a combined sequence of messages, again in timestamp order. If two messages have the same timestamp, then it delivers the messages in increasing order of protocol tag (where the protocol tags are linearly ordered). The MGIOPEngine can deliver a message from a group communication protocol when the Message Ordering Queue contains a message, from each of the other protocols, with a higher timestamp. The group communication protocols must deliver heartbeat messages periodically to prevent excessive delays under light traffic conditions.

7.3 Object/Host Group Table

The MGIOPEngine maintains the correspondence between the object group identifiers, host group identifiers and host group membership.

Each multicast group communication protocol maintains the correspondence between the destination host group identifier, host group membership and destination multicast address.

8 Sending and Receiving Messages

8.1 Sending a Request Message

1. The MGIOPEngine at the client determines which connection, and thus which multicast group communication protocol, to use.
2. The MGIOPEngine constructs the MGIOPEngine header containing the MGIOPEngine_tag, mgiope_version, source_domain_id, source_group_id, dest_domain_id, dest_group_id, and connection_id.
3. The MGIOPEngine determines the timestamp to place on the message, using the force() method if necessary.

4. The MGIOPEngine invokes the `send_message()` method of the MGIOPBridge, passing as a parameter the message with its MGIOP header, the `host_group_id` for the destination host group (which supports both the client and server object groups) and the timestamp.

8.2 Receiving a Request Message

1. The MGIOPEngine at the server (client) extracts the `dest_domain_id` (`source_domain_id`) and the `dest_group_id` (`source_group_id`) from the MGIOP header.
2. The MGIOPEngine uses the `dest_domain_id` and `dest_object_group_id` to determine the internal addressing information for the object replica. The MGIOPEngine consults the Object Group Membership Table using the `dest_object_group_id` in the MGIOP header to obtain the `process_id` for the object replica.
3. The MGIOPEngine enqueues the message for delivery, in timestamp order, to the object replica. It delivers the message when it has received a message with a larger timestamp from each of the other hosts in the host group for the source and destination object groups.

The steps for sending and receiving other types of GIOP messages are similar.

9 Consistent Object/Host Groups

The MGIOPEngine maintains a mapping table that holds the correspondence between the `host_group_ids` and the host membership set for each protocol, as shown in Figure 10. The MGIOPEngine records in this table changes to the host group membership that result from changes to the object group membership, namely,

- Adding a new object group
- Removing an object group
- Adding a new replica to an object group, which may result in the addition of a host to an existing host group
- Removing a replica from an existing object group, which may result in the removal of a host from an existing host group
- Removing a host, and all of the objects on that host from their respective objects group, because a host failed.

10 Consistent Message Delivery

To achieve consistent method execution and therefore consistent states of the replicas, the MGIOPEngine uses times-

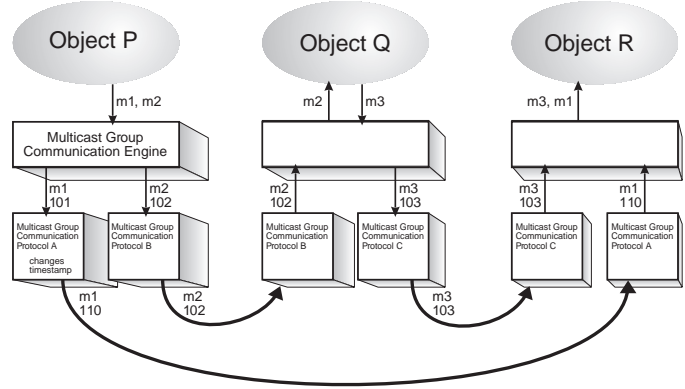


Figure 11: An example that illustrates how, without proper care, causality can be violated when multiple group communication protocols are used.

tamps to ensure consistent message delivery (Lamport causality and total ordering). To ensure Lamport causality and total ordering, the following properties suffice:

1. All messages generated by an object have monotonically increasing timestamps.
2. Any message generated by an object has a timestamp greater than the timestamp of every message delivered to the object prior to the generation of that message.
3. All messages delivered to an object are delivered in the order of increasing timestamps.

To allow flexibility in the group communication protocols, each protocol is allowed to assign timestamps as it sees fit, subject to satisfaction of the above three properties. When passing a message to a protocol for transmission, the MGIOPEngine attaches a timestamp to the message that satisfies these three properties.

We say that a group communication protocol has the *Change Timestamp Property* if it gives a message a timestamp different from that given to the message by the MGIOPEngine. A group communication protocol may increase the timestamp given to a message by the MGIOPEngine, but it must not decrease the timestamp. If a group communication protocol increases the timestamp, it must ensure that it does not violate property 1 for messages that it transmits (which is easy). A group communication protocol delivers messages to the MGIOPEngine in the order of increasing timestamps.

The MGIOPEngine in turn delivers messages in the order of increasing timestamp. To do this, the MGIOPEngine must know, for each object and for each group communication protocol that could deliver a message to that object, a timestamp such that the group communication protocol will not deliver a message with a lower timestamp after it delivers that message. The group communication proto-

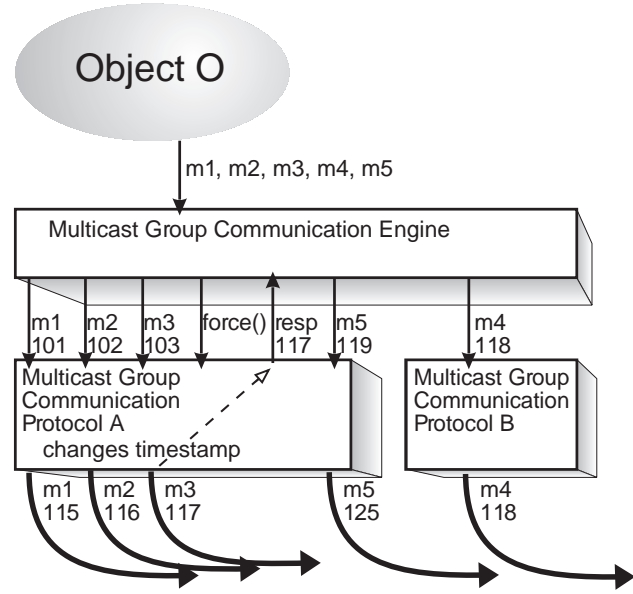
col provides this information in the form of timestamped heartbeat messages. This ensures property 3.

The MGIOPEngine maintains a `current_time` value, which it uses to generate the timestamps of outgoing messages. It compares the timestamps of incoming messages with its `current_time` value. If the `current_time` value is less than the timestamp of an incoming message, the MGIOPEngine increases its `current_time` value by setting it equal to the value of that timestamp. The MGIOPEngine increases the `current_time` value before it uses the `current_time` value as the timestamp of an outgoing message. This ensures property 2. Synchronized clocks are allowed, provided that the synchronization maintains this Lamport clock property.

To ensure property 1, the MGIOPEngine must handle protocols that can increase the timestamps, *i.e.*, have the Change Timestamp Property. Two successive messages generated by the same object using the same protocol cause no problem. Similarly, no problem arises if an object sends a message using a protocol that does not change the timestamps and then sends a message using a different protocol. However, a problem could arise if an object sends a message using a protocol that changes the timestamps and then sends a message using a different protocol. If the timestamp of the first message were increased to be larger than the timestamp of the second message, property 1 would be violated.

Consider the example shown in Figure 11. The MGIOPEngine on the host of object *P* gives message *m1* the timestamp 101 and message *m2* the timestamp 102. Protocol *A* increases the timestamp of message *m1* to 110, while protocol *B* transmits message *m2* with an unchanged timestamp. Object *Q* receives message *m2* and the MGIOPEngine on the host of object *Q* gives message *m3* the timestamp 103 and protocol *C* transmits *m3* with an unchanged timestamp. Object *R* receives message *m3* with timestamp 104 before it receives message *m1* with timestamp 110. This violates Lamport causality because *m1* was originated with a lower timestamp than *m3*.

To address this problem we require that, if an object sends a message using one protocol supported by the MGIOPEngine that has the Change Timestamp Property and then sends a message using a different protocol supported by the MGIOPEngine, the MGIOPEngine invokes a special method, called the `force()` method, on the first protocol before invoking the method of the MGIOPBridge on the second protocol to transmit the second message. The `force()` method returns a timestamp that is greater than or equal to the timestamp chosen by the first protocol for the message it transmitted. The MGIOPEngine updates its `current_time` to be the greater of its `current_time` and the timestamp returned by the `force()` method. The `force()` method may incur a delay before the protocol can determine



1. Initially the MGIOPEngine's `current_time` is 100.
2. The MGIOPEngine passes message *m1* to multicast group communication protocol *A* with timestamp 101. No force is necessary.
3. The MGIOPEngine passes message *m2* to protocol *A* with timestamp 102. No force is necessary.
4. The MGIOPEngine passes message *m3* to protocol *A* with timestamp 103.
5. The MGIOPEngine invokes the `force()` method on protocol *A* and waits for a response.
6. Protocol *A* transmits message *m1* with timestamp 115.
7. Protocol *A* transmits message *m2* with timestamp 116.
8. Protocol *A* transmits message *m3* with timestamp 117.
9. Protocol *A* responds to the `force()` method with timestamp 117.
10. The MGIOPEngine sets its `current_time` to 117.
11. The MGIOPEngine passes message *m4* to multicast group communication protocol *B* with timestamp 118. No force is necessary.
12. The MGIOPEngine passes message *m5* to protocol *A* with timestamp 119.
13. Protocol *B* transmits message *m4* with timestamp 118.
14. Protocol *A* transmits message *m5* with timestamp 125.

Figure 12: An example of the use of the `force()` method. Multicast group communication protocol *A* has the Change Timestamp Property, while multicast group communication protocol *B* does not. Object *O* sends the following messages: *m1*, *m2*, *m3* using protocol *A*, *m4* using protocol *B*, and *m5* using protocol *A*. Without the force method, *m3* would have been transmitted with timestamp 117 and *m4* with timestamp 104, resulting in a violation of Property 1.

that timestamp and, thus, is analogous to a forced write to disk. This ensures property 1.

The requirement that any one message must be handled by a single multicast group communication protocol for all of its destinations implies that the message has the same timestamp at all of its destinations. That requirement, together with Property 2, suffices to ensure delivery in a total order that is consistent system-wide.

11 Related Work

Over the past 15 years, there has been much work on multicast group communication systems [13]. These systems are useful both for fault-tolerant and highly available applications and for groupware and cooperative work applications.

Multicast group communication systems include the Isis, Horus and Ensemble systems [4, 22], which provide the services of multicast, causal multicast and atomic (total order) multicast. Those systems provide increasing flexibility in allowing the user to choose the protocol most appropriate for the application.

The Trans/Total system [12] includes the Trans protocol which provides a causal order on messages, and the Total algorithm which converts this causal order into a total order on messages. The Transis system [2] is based on the Trans protocol and on the Isis application programmer interface. The Totem system [14] uses a ring-based protocol to achieve robust operation and high performance.

The above systems are primarily oriented towards local-area networks. More recent work has focused on the development of group communication systems that are scalable and oriented towards wide-area networks.

The InterGroup system [3] is intended for the Internet, and is designed to support very large groups. It is intended for Distributed Collaboratory Environments, which are intended to allow scientists and engineers to collaborate, and to control equipment remotely, over the Internet. The Atomic Group system [9] is intended for ATM networks, and is designed to support large numbers of small groups. Several other researchers [6, 19, 21] have also undertaken work on multicast group communication systems that aim for scalability and wide-area operation.

Several systems have been developed that use multicast group communication to augment CORBA application objects with high availability and fault tolerance.

These systems include the Electra toolkit, implemented on top of Horus, and Orbix+Isis, implemented on top of the Isis [11]. Both Electra and Orbix+Isis integrate the replication and group communication mechanisms into the ORB and require modification of the ORB.

The Eternal system [15, 16] that we have developed provides fault tolerance for CORBA, using the Totem system to maintain replica consistency in a manner that is transparent to the ORB. In addition to transparency to

the ORB, Eternal has the objectives of transparency to the application and ease of application programming.

The Maestro toolkit adds reliability and high availability to CORBA applications, particularly for applications with unreplicated clients and replicated servers. The AQuA framework [5] uses the Ensemble/Maestro [22, 23] toolkits, as well as the Quality Objects (QuO) runtime and the Proteus dependability property manager, to provide fault tolerance for CORBA.

The Object Group Service (OGS) [7] provides object replication through a set of services implemented on top of the ORB, including a group service, a consensus service, a monitoring service and a messaging service.

The Distributed Communicating Object group system (DCO) [8] also provides for communicating object groups, reliable communication protocols, group membership and management. The goals of DCO are similar to those of the Eternal system.

Other systems have been developed that use multicast group communication for groupware and computer-supported cooperative work (CSCW).

The Java Collaborative Environment (JCE) [1] allows a group of Internet users to share single-user Java applications for synchronous collaboration. JCE is a replicated tool architecture in which each participant runs a copy of the application and the activity of each user is multicast to all of the conference participants.

The mStar environment [18] provides support for scalable distributed teamwork. It enables collaborative reviewing of text and images, text based group chat, distributed voting and shared WWW objects, based on IP-multicast.

The system of Sun, Zhang and Yang [20] synchronizes distributed group operations in order to maintain consistency of shared documents in cooperative editing environments.

12 Conclusion

We have shown how CORBA can be enhanced with multicast group communication capabilities, and have given a concrete mapping of CORBA's General Inter-ORB Protocol specification onto a multicast group communication protocol. We have presented a multicast group communication engine and bridge for CORBA that allows the use of multiple group communication protocols concurrently, with different implementation strategies appropriate to different environments, such as local-area vs. wide-area networks and Internet vs. ATM. The multicast group communication engine timestamps messages using Lamport timestamps, and delivers messages in timestamp order. When switching from one protocol to another, where the second protocol increases the timestamps of the messages, the multicast group communication engine employs a force() method to ensure Lamport causality and total ordering.

References

- [1] H. Abdel-Wahab, B. Kvande, O. Kim and J. P. Favreau, "An Internet collaborative environment for sharing Java applications," *Proceedings of the IEEE 6th Workshop on Future Trends of Distributed Computing Systems*, Tunis, Tunisia (October 1997), pp. 112-117.
- [2] Y. Amir, D. Dolev, S. Kramer and D. Malki, "Transis: A communication sub-system for high availability," *Proceedings of the IEEE 22nd Annual International Symposium on Fault-Tolerant Computing*, Boston, MA (July 1992), pp. 76-84.
- [3] K. Berket, L. E. Moser and P. M. Melliar-Smith, "The InterGroup protocols: Scalable group communication for the Internet," *Proceedings of the IEEE Global Telecommunications Conference GLOBECOM, Global Internet '98 Mini-Conference Record*, Sydney, Australia (November 1998), pp. 100-105.
- [4] K. P. Birman and R. van Renesse, *Reliable Distributed Computing with the Isis Toolkit*, IEEE Computer Society Press, Los Alamitos, CA, 1994.
- [5] M. Cukier, J. Ren, C. Sabnis, W. H. Sanders, D. E. Bakken, M. E. Berman, D. A. Karr and R. E. Schantz, "AQuA: An adaptive architecture that provides dependable distributed objects," *Proceedings of the IEEE 17th Symposium on Reliable Distributed Systems*, West Lafayette, IN (October 1998), pp. 245-253.
- [6] P. Ezhilchelvan, R. Macedo and S. Shrivastava, "Newtop: A fault-tolerant group communication system," *Proceedings of the IEEE 15th International Conference on Distributed Computer Systems*, Vancouver, Canada (May 1995), pp. 296-306.
- [7] P. Felber, B. Garbinato and R. Guerraoui, "The design of a CORBA group communication service," *Proceedings of the IEEE 15th Symposium on Reliable Distributed Systems*, Niagra-on-the-Lake, Ontario, Canada (October 1996) pp. 150-159.
- [8] W. Jia, "Communicating object group and protocols for distributed systems," *Journal of Systems and Software* (March 1999), vol. 45, no. 2, pp. 113-126.
- [9] R. R. Koch, L. E. Moser and P. M. Melliar-Smith, "The Atomic Group protocols: Group communication protocols for ATM networks," in preparation.
- [10] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, no. 7 (July 1978), pp. 558-565.
- [11] S. Landis and S. Maffei, "Building reliable distributed systems with CORBA," *Theory and Practice of Object Systems*, vol. 3, no. 1, (April 1997), pp. 31-43.
- [12] P. M. Melliar-Smith, L. E. Moser and V. Agrawala, "Broadcast protocols for distributed systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 1 (January 1990), pp. 17-25.
- [13] P. M. Melliar-Smith and L. E. Moser, "Group communication," *Encyclopedia of Electrical and Electronics Engineering*, Vol. 8, ed. J. G. Webster, John Wiley & Sons (February 1999), pp. 500-512.
- [14] L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, R. K. Budhia and C. A. Lingley-Papadopoulos, "Totem: A fault-tolerant multicast group communication system," *Communications of the ACM*, vol. 39, no. 4 (April 1996), pp. 54-63.
- [15] L. E. Moser, P. M. Melliar-Smith and P. Narasimhan, "Consistent object replication in the Eternal system," *Theory and Practice of Object Systems*, vol. 4, no. 2 (1998), pp. 81-92.
- [16] P. Narasimhan, L. E. Moser and P. M. Melliar-Smith, "Replica consistency of CORBA objects in partitionable distributed systems," *Distributed Systems Engineering*, vol. 4, no. 3 (September 1997), pp. 139-150.
- [17] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, Revision 2.2, OMG Technical Document formal/98-07-01 (February 1998).
- [18] P. Parnes, K. Synnes and D. Schefstrom, "The CDT mStar environment: Scalable distributed teamwork in action," *Proceedings of the International ACM SIGGROUP Conference on Supporting Group Work: The Integration Challenge*, Phoenix, AZ (November 1997), pp. 167-176.
- [19] L. E. T. Rodrigues, H. Fonseca and P. Verissimo, "Totally ordered multicast in large-scale systems," *Proceedings of the IEEE 16th International Conference on Distributed Computing Systems*, Hong Kong (May 1996), pp. 502-510.
- [20] C. Sun, Y. Zhang and Y. Yang, "Distributed synchronization of group operations in cooperative editing environments," *Concurrent Engineering: Research and Applications*, vol. 4, no. 3 (September 1996), pp. 293-302.
- [21] T. Tachikawa, H. Higaki, M. Takizawa, M. Gerla *et al*, "Flexible wide-area group communication protocols -- international experiments," *Proceedings of the 1998 ICPP Workshop on Architectural and OS Support for Multimedia Applications*, Minneapolis, MN (August 1998), pp. 105-112.
- [22] R. van Renesse, K. P. Birman, M. Hayden, A. Vaysburd *et al*, "Building adaptive systems using Ensemble," *Software - Practice and Experience*, vol. 28, no. 9 (July 1998), pp. 963-979.
- [23] A. Vaysburd and K. Birman, "The Maestro approach to building reliable interoperable distributed applications with multiple execution styles," *Theory and Practice of Object Systems*, vol. 4, no. 2 (1998), pp. 73-80.